


# Práctica 4. ESTRUCTURAS DE CONTROL ITERATIVAS

 2 Sesiones

Semanas: del 15 y 22 de Octubre 

## OBJETIVO:

- ✓ Conocer y manejar con habilidad los distintos tipos de estructuras de control iterativas

*No olvides que antes de pasar a la fase de diseño deberemos **analizar** exhaustivamente **el problema***

Las **sentencias de control iterativas** son aquellas que nos permiten variar o alterar la secuencia normal de ejecución de un programa haciendo posible que un grupo de acciones se ejecute más de una vez de forma consecutiva. Este tipo de instrucciones también recibe el nombre de **bucle**.

En los bucles se suelen utilizar algunas variables para unas tareas específicas, tales como **contadores, acumuladores o interruptores**.

### - Contadores:

Un contador no es más que una variable destinada a contener un valor que se irá incrementando o decrementando en una cantidad fija. Se suelen utilizar para el control de procesos repetitivos.

### - Acumuladores:

Un acumulador es una variable destinada a contener distintas cantidades provenientes de los resultados obtenidos en operaciones aritméticas previamente realizadas de manera sucesiva, lo que nos permitirá obtener el total acumulado de dichas cantidades. A diferencia de los contadores, no controlan los procesos repetitivos. Su inicialización depende de en qué operación matemática van a ser utilizados.

### - Interruptores (switches):

Los interruptores, también denominados conmutadores o indicadores, son variables que pueden tomar dos únicos valores considerados como lógicos y opuestos entre sí a lo largo de todo el programa (0 o 1, 1 o -1, Verdadero o Falso, on/off, etc.).

Su objetivo es recordar en un determinado lugar del programa una ocurrencia o suceso acaecido o no con antelación, o hacer que dos acciones diferentes se ejecuten alternativamente en un proceso repetitivo. También deben ser inicializados. No se debe abusar de su utilización cuando no sea necesario.

**Ejercicio Resuelto 1.** Implementa un programa que lea dos números naturales desde teclado. A continuación el programa debe contar y sumar los números que hay entre ellos. Por último debe imprimir en pantalla esa información.

```
#include <iostream>
using namespace std;

main () {
    int i, cont, suma, num1, num2, aux;

    cout << "Introduce un número:";
    cin >> num1;
    cout << "Introduce otro número:";
    cin >> num2;
    if (num1>num2){
        aux=num1;
        num1=num2;
        num2=aux;
    }
    cont=0; suma=0;
    for (i=num1+1; i<num2; i++){
        cont++;
        suma = suma+i;
    }
    cout << "Hay " << cont << " números entre el " << num1 << " y el " << num2;
    cout << "\nLa suma es " << suma;
}
```

**Ejercicio Resuelto 2.** Implementa un programa que lea un número natural por teclado y calcule y visualice el número formado por las mismas cifras pero en sentido inverso.

```
#include <iostream>
using namespace std;

main () {
    int num;

    do {
        cout << "Introduce un número natural: ";
        cin >> num;
    } while (num <= 0);

    cout << "El número con las cifras en orden inverso es: ";
    while (num > 0) {
        cout << num%10;
        num = num/10;
    }
}
```

**Ejercicio Resuelto 3.** Implementa un algoritmo que visualice por pantalla la siguiente figura, preguntando al usuario el número de estrellas que tiene que contener la línea final.

```
*
**
***
****
*****
*****
```

```
#include <iostream>
using namespace std;
main() {
    int n, fil, col;

    cout <<"Introduce el número de estrellas de la línea final:";
    cin >> n;
    for(fil=1; fil<=n; fil++){
        for(col=0; col<fil; col++){
            cout << '*';
        }
        cout << endl;    //fin de linea
    }
}
```

```
}
}
```

**Ejercicio Propuesto 1. Implementa un algoritmo que lea un conjunto de números reales hasta que se introduzca el valor cero y, a continuación, calcule y visualice la media y la varianza de los números introducidos.**

**Ejemplo:**

```
Introduce un número --0 para finalizar--: 2
Introduce un número --0 para finalizar--: 3.5
Introduce un número --0 para finalizar--: 6
Introduce un número --0 para finalizar--: 2.1
Introduce un número --0 para finalizar--: 7
Introduce un número --0 para finalizar--: 0
```

```
La media es 4.12
La varianza es 4.1576
```

Recuerda: La media y la varianza de un conjunto de números  $x_1, x_2, \dots, x_n$  se calcula de la siguiente manera:

$$media = \frac{\sum_{i=1}^n x_i}{n} \quad \quad \quad varianza = \frac{\sum_{i=1}^n x_i^2}{n} - media^2$$

**Ejercicio Propuesto 2. Implementa un programa que lea un número natural desde teclado. A continuación el programa debe calcular el siguiente número primo, mayor que el introducido, y mostrarlo por pantalla.**

**Ejemplo:**

```
Introduce un número natural:
-2
Ese número no es natural.

Introduce un número natural:
14
El siguiente primo es 17
```

**Ejercicio Propuesto 3. Implementa un programa que represente gráficamente un polinomio de tercer grado  $f(x) = ax^3 + bx^2 + cx + d$ , de forma invertida (es decir, los valores de  $x$  en el eje vertical y los de  $f(x)$  en el horizontal). Para ello el programa solicitará en primer lugar el número de valores que se quieren representar (empezando en 0) y el valor de cada uno de los cuatro coeficientes del polinomio, que serán enteros. El programa debe controlar que el número de valores y los coeficientes sean todos mayores o iguales que 0.**



**Ejemplo: representar el polinomio  $x^2 + 4$**

```
Introduce número de valores: 6
Introduce coeficiente a: 0
Introduce coeficiente b: 1
Introduce coeficiente c: 0
Introduce coeficiente d: 4
```

```
0 |      *
1 |     *
2 |    *
3 |   *
4 |  *
5 | *
```

**Ejemplo: representar el polinomio  $x^3 + 3x$**

```
Introduce número de valores: 5
Introduce coeficiente a: 1
Introduce coeficiente b: 0
Introduce coeficiente c: 3
```

Introduce coeficiente d: 0

```

0 | *
1 |   *
2 |       *
3 |           *
4 |               *
```

**Ejercicio Propuesto 4.** Implementa un programa que calcule el número pi con una precisión de  $n$  decimales, siendo  $n$  un valor introducido por teclado. Puedes utilizar el algoritmo de Gauss-Legendre, que consiste en lo siguiente:

dados los siguientes valores iniciales:

$a_0 = 1.0$   
 $b_0 = 1.0 / \sqrt{2}$   
 $t_0 = 1.0 / 4$   
 $p_0 = 1.0$

a partir de estos valores iniciales, realizamos sucesivamente las siguientes operaciones:

$a_{n+1} = (a_n + b_n) / 2.0$   
 $b_{n+1} = \sqrt{a_n * b_n}$   
 $t_{n+1} = t_n - p_n * \text{pow}(a_n - a_{n+1}, 2)$   
 $p_{n+1} = 2 * p_n$

finalmente el número pi se aproxima de la siguiente forma:

$\text{pi} = \text{pow}(a_n * b_n, 2) / (4 * t_n)$

**Ejemplo:**

¿Cuántos decimales quieres del número pi?

8

Aproximación del número pi con 8 decimales : 3.14159265

**Aclaraciones:**

- $\text{sqrt}(num)$  y  $\text{pow}(a,b)$  son funciones predefinidas del lenguaje C:  $\text{sqrt}(num)$  calcula la raíz cuadrada de  $num$  y la función  $\text{pow}(a,b)$  calcula la potencia de base  $a$  y exponente  $b$ . El concepto de función lo veremos más adelante en la asignatura. De momento utiliza las llamadas a esa función tal y como se ilustra en estas fórmulas. Es necesario además que incluyas la librería de C que contiene la definición de estas funciones: `#include <cmath>`
- Para imprimir un número real con un número  $n$  de decimales, utiliza las siguientes sentencias antes de la orden 'cout' que imprima el resultado de la aproximación del número pi:  
`cout.setf(ios::fixed);`  
`cout.precision(n);`

**Ejercicio Propuesto 5.** Implementa un programa que visualice en pantalla cada una de las siguientes figuras, preguntando al usuario el número de filas  $n$  que deben tener las figuras (En el ejemplo mostrado,  $n = 6$ ). Ten en cuenta que la solución que propongamos debe contener sentencias de salida por pantalla en las que se impriman exclusivamente uno solo de los siguientes caracteres: blanco ' ', asterisco '\*' y letra 'o'.

```

*****
*o o o*
*o o*
*o*
**
*
```

```

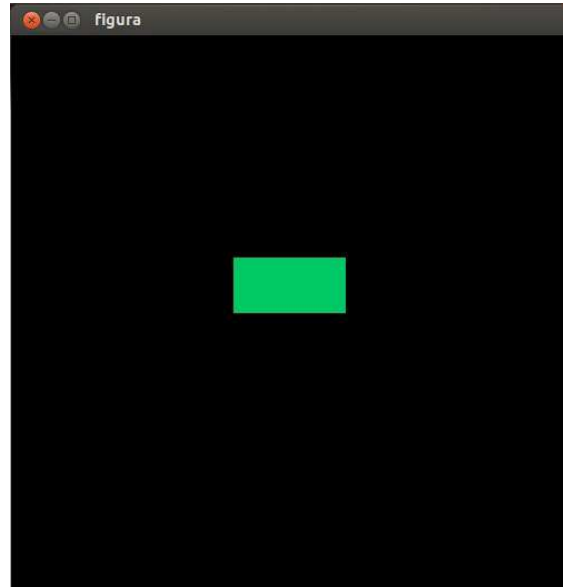
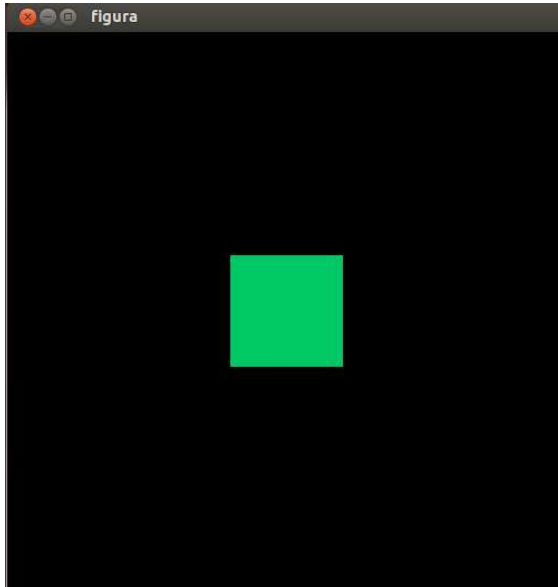
o*****o
*****
****
***
**
o
```

```

      *
    *o*
  *o*o*
*o*o*o*
*o*o*o*o*
*o*o*o*o*o*
```

**Recomendación:** Resuelve primero el ejercicio sin distinguir en las figuras los caracteres '\*' y 'o', es decir, dibuja sólo cada figura con el carácter '\*'. Después modifica tu solución inicial para incluir también en las figuras el carácter 'o'.

**Ejercicio Propuesto 6.** Basándote en el ejercicio propuesto 9 de la práctica 3 (dibujar un cuadrado o un rectángulo), escribe un programa que dibuje el cuadrado o el rectángulo relleno empleando la librería gráfica gfx.



## ERRORES DE PROGRAMACIÓN COMUNES

- **Bucles infinitos.** A veces no se controla bien la condición de control y se ejecuta un bucle infinito.

Por ejemplo:

```
contador = 5;
while (contador != 0) {
    contador = contador - 2;
}
```

La variable contador va tomando los valores 5, 3, 1, -1,... y estaríamos en un bucle infinito, puesto que nunca llega a tomar el valor 0.

- **Uso de la misma variable de control en bucles anidados.** Habitualmente, cuando se programa se usan variables del tipo i, j, k... para controlar bucles anidados. En bucles anidados es frecuente que se olvide la variable de control y se reutilice una ya usada en bucles más externos o usada para otro propósito. Para evitar esto, conviene reservar ese tipo de variables exclusivamente para el control de bucles y no usar variables poco descriptivas (como *contador*) para controlar los bucles.

- **Suponer que el compilador asigna valor cero a las variables sin inicializar.** Si se escribe:

```
int i;
while (i < 10) ...
```

se está asumiendo que el valor inicial de i es 0. Esto no tiene por qué ser cierto. Nunca se deben hacer ese tipo de suposiciones y se deben inicializar todas las variables.

- **Errores de tipo fallo por uno en número de iteraciones.** Si se quiere ejecutar un bucle controlado por contador *n* veces, se puede hacer lo siguiente:

```
contador = 1;
while (contador < n) {
    contador ++;
}
```

Sin embargo, este bucle sólo se ejecuta *n-1* veces. Imagine que escribe:

```
contador = 0;
```

```
while (contador <= n) {  
    contador ++;  
}
```

Entonces se ejecuta  $n+1$  veces. Evite este tipo de errores fuera por uno. Son fáciles de cometer y muy difíciles de detectar. ¿Qué debería escribir? Lo siguiente:

```
contador = 0;  
while (contador < n) {  
    contador ++;  
}
```

O también sería correcto:

```
contador = 1;  
while (contador <= n) {  
    contador ++;  
}
```

➤ **Confundir el criterio de comparación del *for*.** Si se escribe:

```
for (i=0; i < 10; i++) ...
```

se ejecuta el contenido del bucle 10 veces. Si en lugar de eso se escribe:

```
for (i=0; i == 10; i++) ...
```

no ocurre lo mismo. ¿Por qué? Pues porque se ejecuta la primera iteración y se compara la condición de control ( $i == 10$ ). Como no es cierta, se termina el bucle.